ESD-TR-68-61

AL 61139

ESLL

Semiannual Technical Summary

Graphics

1 of 1

31 May 1968

Prepared for the Advanced Research Projects Agency
under Electronic Systems Division Contract AF 19(628)-5167 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Lexington, Massachusetts

AD0671125

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, this work was supported by the U.S. Advanced Research Projects Agency of the Department of Defense under Air Force Contract AF 19(628)-5167 (ARPA Order 691).

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

GRAPHICS

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
ADVANCED RESEARCH PROJECTS AGENCY

1 DECEMBER 1967 – 31 MAY 1968

ISSUED 19 JUNE 1968

LEXINGTON MASSACHUSETTS

SUMMARY

The LEAP system has been modified to provide for merging current and previously saved data structures and incorporation of a sublanguage for communicating with the interrupt executive. Breakpoint trapping facilities have been used for running programs in single-step mode, program-timing experiments, and in providing variable response delays for human factors studies.

AMBIT/G, a programming language for manipulating directed graphs, is being implemented using LEAP and the interactive graphics facilities of TX-2.

A first set of working semiconductor circuits has been made from masks generated via TX-2 programs. A written-input version of the mask-generation program has been used to produce masks for a Read-Only Memory design. In addition, the laboratory service facility is now using this program for almost all of its hybrid circuit mask layout jobs.

Work has continued on computer-aided circuit testing with the addition of new features to the TIC terminal and the development of procedures for diagnosing single-gate failures in complex arrays.

Experiments are planned for evaluating a TV display buffered by the film memory soon to be installed in TX-2. Modifications to the conic generator will be made shortly which should eliminate the generator limitations on drawing speed.

Further theoretical work continues on constraint systems and the application of homomorphic filtering to picture processing.

Accepted for the Air Force
Franklin C. Hudson
Chief, Lincoln Laboratory Office

CONTENTS

GLOSSARY

AMBIT/G	Graphical programming language for manipulation of directed graphs
APEX	TX-2 time-sharing executive
BMRS	Ballistic missile re-entry system
LDP	Link departure point
LEAP	Extension of LABGOL, including associative language
LSI	Large-scale integrated circuit technology
TIC	Testing integrated circuits
VITAL	A compiler-compiler system

GRAPHICS

I. LANGUAGES

A. LEAP System

The continued evolution of the LEAP system over the past several months has been guided by several new needs from its users. Two major additions to the system have been made:

(1) A facility for merging a working data structure with a previously saved data structure.

(2) A simple sublanguage for communicating with the part of the time-sharing system which handles input interrupts.

The mechanism for merging LEAP data structures is quite straightforward: all ITEMs in the saved structure except those declared with no data-type will be added as new ITEMs to the working data structure, and all TRIPLEs in the saved structure will be adjusted to preserve the relations between these new ITEMs and then will be added to the working data structure.

The new interrupt sublanguage allows LEAP programs to "activate" the various input devices at a console, thereby asking the time-sharing executive to gather the relevant information whenever an input event occurs, and report this information to the user when he is next active. The user may ask for certain status information to be recorded along with the specified input event. For example, he may ask that the real-time clock reading be recorded whenever a knob changes state:

(1) <u>activate</u> <u>βknobs</u> <u>reporting</u> <u>βrtc;</u>*

The executive reports input information to the user by maintaining a list of events, each with appropriate cause and status information. The user may ask for information about the next event; an entry will be removed from the list of events, and the cause and status information will be reported to him. If the list is empty, he will be notified. Typically, the user calls a reserved procedure to get information about the next event:

(2) <u>getnextint;</u>

If the next event were a knob change in state, this procedure would store the appropriate code-word into the reserved variable <u>αcause</u>, and the state of the four knobs into the reserved variables <u>αknob 1</u>, <u>αknob 2</u>, <u>αknob 3</u>, and <u>αknob 4</u>. If a request to report the real-time clock reading accompanied the knob activation statement (as in (1)), the reading taken at the time of the event would be stored into the reserved variable <u>αrtc</u>. If the list of events is empty, the getnextint procedure would store zero into <u>αcause</u> and then return.

A user program may request that it be inactivated and awakened when the next input event occurs:

* Reserved words in the language are underlined.

(3) shade;

A typical program fragment might be the following (the code for a knob event is 7):

```
NEXT ←          getnextint;
                if αcause = 0 then
                  begin
                        shade;
                     goto   NEXT
                   end;
                if αcause ≠ 7 then ERROR:
                  .
                  .
                  .
                goto    NEXT;
```

B. On-Line Trapping and Debugging

Development has continued on the implementation and experimentation with new debugging techniques in the Mark 5 assembler system. Many of the debugging techniques depend very heavily on the mechanism for overlaying virtual memories, or maps as they are called, in the APEX time-sharing system.

A map is a virtual memory containing files and including a set of live registers, index registers, and a program counter. A user is provided with a stack of maps for running programs; at any time the stack can be pushed down by going to a program in a new map and then popped when the job in the new map is completed. Of particular interest for debugging is the debugging "ghost map" which is created whenever a trap or error occurs. Thus, the debugger is called in and pushes down the map stack when it is needed to field a trap or error; it goes away when the user wishes to resume his object program. Since the debugger and the object program are in different maps, conflicts over core requirements and possible destruction of one by the other are avoided.

The framework provided by maps has made possible a simple approach to the implementation of an automatic command executor for breakpoint trapping. As provided by Mark 5, a user may designate a file which contains commands to be executed whenever a trap occurs. At trap time the commands are analyzed. The Mark 5 commands are performed in the debug ghost map; the others are sent off to the appropriate programs in higher maps. The most common commands are those for printing out the information about the state of the object program. One may thus obtain an automatic printout showing the history of the values of a variable, the state of live registers at various places in a program, etc.

Several deficiencies presently exist in this automatic command facility. Only one file may be designated for all the breakpoints, rather than a different file for each breakpoint. Also, the lack of conditional statements does not permit the output to be varied according to conditions in the object program.

Using the breakpoint trapping capabilities provided by the TX-2 hardware, a single-stepping mode for running programs has been created. In this mode the object program is trapped after each instruction. When the mode is combined with automatic command execution, the behavior of a routine and its data may be observed on a microscopic level. Single-stepping is of use primarily in desperate situations; at such times it is of incomparable value.

Again using breakpoint trapping, a simple program-timing facility has been developed. To time a program, a user specifies breakpoints, indicates that he desires timing, and runs his program. When the breaks occur, the contents of the TX-2 real-time clock are recorded. The difference between two successive clock values is the elapsed running time between the corresponding traps.

A novel use of breakpoint trapping has recently been devised to support a human-factors project. The goal of the project is to study the effect on problem solving of different delays in response from a computer. To achieve the delay, breakpoint traps are placed on the sections of the APEX time-sharing system which initiate output. When the trap occurs, the "delayer" fielding the trap in the debug ghost map waits the desired amount of time. Then it resumes the output routine, this time with no breakpoint trapping specified.

II. GRAPHICS AND APPLICATIONS

A. AMBIT/G

1. Introduction

AMBIT/G is a graphical programming language, developed by Christensen,[1] for the manipulation of directed graphs. The data upon which an AMBIT/G program operates is a two-dimensional network of nodes and directed linkages between nodes. Each node in this data graph (see Fig. 1) has an associated shape, and each shape has a number of special points from which linkages are allowed to emanate. Each of these link departure points (LDP's) may have no more than one departing link. A node may have an associated string of alphanumeric characters (a name).

An AMBIT/G program consists of a number of statements in a 2-D graphical form, each specifying:

(a) A subgraph to find in the data graph.

(b) Changes to make to the linkages in the data graph if the subgraph is found.

(c) The statement to execute next if the subgraph is found.

(d) The statement to execute next if the subgraph is not found.

The AMBIT/G language affords an easy and natural way to express algorithms for building and pruning tree structures, garbage collecting a list-structure, maintaining a free-storage allocation facility, etc. Typically, a user's natural view of such data structures and structure manipulation algorithms corresponds with the graphical representations of these in the AMBIT/G language.

The AMBIT/G system avoids searching for specified subgraphs in the data graph by requiring that all source nodes in the subgraph have specified names. Since each named node corresponds to exactly one node in the data graph, and the system remembers where these are, the subgraph matching procedure begins at a known place in the data graph, and works downstream. This implies that one cannot ask for those nodes which point to a specified node.

A preliminary version of an AMBIT/G system has been implemented on the TX-2 computer. The available facilities on the TX-2 for interactive graphics are well suited to such an implementation.

The overall goal in this work is the creation of an experimental, user-oriented AMBIT/G system. Flexibility is of primary importance; modifications and extensions to the AMBIT/G system should be easy to make. For this reason, the system is entirely written in the LEAP language, which has high-level facilities for doing interactive input, display output, and data structure manipulations. The system is composed of three programs, which took a total of eight man-weeks to write and debug:

(a) An input analyzer and control monitor.
(b) An AMBIT/G program interpreter.
(c) A simple data graph output program.

2. A User's View of the AMBIT/G System

Typically, a user of AMBIT/G sits down in front of the computer console and proceeds to specify his node shapes, program statements, and initial data graph by using the tablet stylus to draw and point. He works in one of four modes, which are specified by light targets at the right of the working area:

(a)	SHAPES	(draw in or edit a node shape declaration).
(b)	DATA	(draw in or edit the initial data graph).
(c)	PROGRAM	(draw in or edit a program statement).
(d)	DEBUG	(specify which part of the data graph to look at).

The node shapes and names that he has declared appear to the left of the working area, and he may point to one of these. The working area is shown in Fig. 2. If, while in the DATA or PROGRAM mode, he draws a small x somewhere in the working area, an instance of the last node shape or named node shape to which he pointed will appear where the x was drawn. If he draws an x while in the SHAPES mode, a small LDP indicator will appear, and will become a part of the current node shape declaration. If he draws an x while in the DEBUG mode, an instance of the last named node to which he pointed will appear, and he will be able to point to one of its LDP's, thus asking for a display of that linkage in the data graph.

The user may draw a line, which will be smoothed and then displayed with an arrowhead. In PROGRAM mode, he may draw a line with a loop (-e), and a double line with an arrowhead will be displayed. Single lines (indicating linkages between node shapes) specify linkages in the subgraph to be found in the data graph. Double lines indicate new connections to be made if the subgraph is found.

The user may erase any entity (instance of a node shape, line endpoint, line, LDP) in the working area by superimposing a "scrub mark": UUUU. He may move an entity in the working area by either circling it and then drawing a line to where it should move, or by pointing to the MOVE light target, and then dragging the entity to its new position with the tablet stylus. Linkage lines stretch and contract as appropriate.

Pointing at the FILE light target causes the current node-shape declaration, data graph, or program statement to be saved and filed away for future reference. Only the graphical information is saved; the linkage information is generated from the graphical information in a separate step prior to AMBIT/G program execution.

The user may save his work between sessions on the computer: he may give it a name and write it out on his storage area by pointing to the WRITE light target and then typing in the name; he may read in a named AMBIT/G program by pointing to the READ light target and then typing in the name.

While in the DATA or PROGRAM mode, the user may ask for a two-dimensional syntax check to be performed on the data graph or program statement in the working area by pointing to the CHECK light target. A check is made for dangling links, illegal link sequences, and unreachable nodes, and error indicators are displayed at the appropriate places in the working area (Fig. 3).

When the user finishes drawing in his shapes, program statements, and initial data graph, he points to the ACCEPT light target, which causes a final syntax check to be made on all of his input, and the linkage information for the initial data graph and all program statements to be generated. Errors are detected and reported. After his work is accepted successfully, he may point to the GO light target, which causes his program to be executed. Facilities are available in the DEBUG mode for interrupting execution at a specified program statement, analyzing the state of the data graph, and then resuming execution.

3. Interesting Aspects of the Design

Several interesting design decisions were made after implementing and experimenting with various alternative designs; convenience for the user was the primary consideration in these decisions.

No push-buttons or toggle switches are used since it was felt that these are confusing and distracting. Use of a simple drawn character or a light target is easier.

The user does not draw node shape instances, but rather points to the desired shape, and then draws an x at the desired position. The special symbols that the user may draw are simple enough to recognize easily, and are quite convenient to use.

There are two ways to move an entity in the working area: by drawing a "move character," or by entering "move mode" and dragging the entity with the tablet stylus. Experience has indicated the necessity for both. The move character is useful for a single move, where the destination is clear. The move mode is useful if many moves are to be performed, or if the user wants feedback as the entity moves.

When the user draws in the working area, he is drawing on a square grid. That is, the endpoints of all lines and the centers of all node instances will automatically be placed on the nearest grid point. This feature makes neat drawings, and does not restrict the user appreciably. He may be reasonably sloppy in drawing a link, for example, and the startpoint will be made to coincide with a nearby LDP. There are two grid sizes· one for the SHAPES mode, and one for the other modes. When drawing in a SHAPE declaration, it is convenient to work on a fairly gross grid (the working area has 16 grid points on a side in the SHAPES mode; in the other modes, it has 64 grid points on a side).

No AMBIT/G dependent graphical syntax check is made as the user is drawing in the working space. The syntax of his drawn data graph and program statements is checked only upon explicit request from him, or at ACCEPT time. This allows him to file away a partially completed picture or data graph. The system is considerably simplified by this feature, because it needs only to remember picture parts and their positions until ACCEPT time. Also a change made to a node shape declaration has no subtle effect on the connectivity of a program statement in which there is an instance, since no connectivity information is kept.

4. Conclusion

The preliminary AMBIT/G system has been used to implement two examples:

(a) The list-structure garbage collection program used as the example in Ref. 1.

(b) A reductions-analysis program for parsing an input string from a simple grammar and building the computation tree.

The experience of using the AMBIT/G system in time-sharing on TX-2 has pointed out some major inadequacies in the environment. The AMBIT/G system is fairly large, and typically requires fast response and a small time slice. When the time-sharing system has a medium to heavy load, response lag time increases to 10 to 20 seconds. Typically, the user must wait about 15 seconds after drawing a symbol (or pointing to a light target) before he can draw the next symbol. For this kind of application on TX-2 either the environment in time-sharing must be reorganized or the machine must be used in dedicated mode.

The poor response in time-sharing is caused primarily by the need to swap users in and out of core. The response lag time increases dramatically as soon as the total active user core requirement exceeds available core. This problem is aggravated by two things:

(a) The swapping mechanism being used is very slow (it is a FASTRAND II drum, which is meant to be used for file storage).

(b) People tend to write very large programs (the Reckoner is a notable exception) because there is no convenient way to segment programs. For example, the LEAP compiler does not compile relocatable code, nor will it compile subroutines separately from a main program. There is no relocatable loader facility; even the assembler assembles non-relocatable code.

Other factors which contribute to system overhead are:

(a) The main frame is used to process interrupts and track the two tablets. If both tablets are active, this overhead is large.

(b) The display structures for all active displays reside in core; the display generator steals memory cycles to refresh the displays.

In the course of this work, several ideas for extensions to the AMBIT/G language have arisen. For example, a method for building subroutines is needed, and a facility for manipulating algebraic values is desirable. Other suggestions include the following:

(a) Allow the display of several program statements at once, showing the control flow linkages between them.

(b) Allow the user to specify in his program the portion of the output data graph which is to be displayed (some work has begun on this).

(c) Display a selected portion of the data graph dynamically, as the program is executed.

(d) Use the new character recognition program on TX-2 to recognize the special drawn characters, and drawn alphanumeric characters (eliminate the typewriter input for names).

The programming work for the AMBIT/G system was done entirely in the LEAP language. The facilities in LEAP for expressing interactive input and display output were found to be very powerful, and very easy to use. The ability to make changes to the programs easily made much of the experimentation feasible; a program written in LEAP is relatively easy to read and understand.

The language forms for building and manipulating a data structure in LEAP were used extensively to create and process the internal representation of the AMBIT/G data graph and program statements. It was unnecessary to design and implement an elaborate list structure to house the internal representation; we were free to concentrate on the design of the system because we were insulated from many of the intricate details of the data structure implementation. Inefficiencies have been introduced into the AMBIT/G system by the use of LEAP; these are noticeable only when a large computation has to be made (when checking, accepting, or interpreting). The system could be made considerably smaller and faster if it were recoded in machine language, but this would require much work, and would remove much flexibility from the system.

B. Semiconductor Mask Design

The first working circuits have been made from masks generated via TX-2 programs. A simple three-input gate was sketched on the computer display, and tapes punched for a precision pattern generator from the circuit geometry data in the computer. The master reticles from the pattern generator were then reduced and stepped to make the set of masks necessary for fabrication of the circuit on a silicon wafer. The photographs in Fig. 4 show several steps in this design process. Two hours were needed on the computer to settle on the particular layout, and thirty minutes of pattern generator time produced the master reticles. The circuit chosen for fabrication is particularly simple to aid electrical testing of the resulting chips. The results of this first fabrication experiment are most promising, and more complex circuits are being designed. The actual circuit fabrication is done in conjunction with another laboratory research program.

In an interesting and unforeseen way, most recent usage of the layout program has been for trial designs which will never be fabricated. In deciding on a good layout there are many intangibles which a designer must consider, such as questions of supply voltage access, power dissipation, and external connection possibilities. The computer program has been used regularly to try out new ideas, most of which do not work out. It is so much easier to sketch trial layouts on the display and have components placed correctly with the proper spacings and sizes, that the circuit people are now reluctant to take up pencil and paper. Hidden implication becomes rapidly apparent when real geometries can be used easily and changed.

Previous comments on the need for evolution in design programs are still valid. The successful experience to date has indicated that another major revision is needed, and when time permits, the layout program will be completely redone. The desire of the designers to work on even bigger circuits has outmoded some of the simple assumptions of the early programs and complexities of windowing and picture translation must now be added. In addition, the existing program will be restructured to take advantage of some new capabilities in the TX-2 interrupt executive and the LEAP language. Improved operation in time-sharing should result.

An alternative method for pattern description and generation has been developed. In many cases of repetitive patterns, a written description of the pattern is convenient. A system for producing tapes for the precision pattern generator from written pattern descriptions has been developed. This system consists of a host program (written in LEAP) which is augmented with each individual pattern description. The host program contains the basic definitions used for the pattern descriptions, the on-line control and output facilities, and the tape punching operations. An augmented host LEAP program is compiled and then run to obtain first a display of the pattern on the scope, and then, if satisfactory, a punched tape for the precision pattern generator. Thus, the pattern description is itself a LEAP program segment and can be descriptive or generative in nature. All the power of the language is available for creating simple descriptions of complex patterns. The "FOR statement" makes pattern repetitions natural and easy. Coordinated sets of masks can easily be described using a level-indicating variable and conditional statements to control the inclusion of components at each level.

Using this facility, the masks for a 256-bit read-only memory were described, observed, and modified until a satisfactory design was obtained in the span of a few hours. Figure 5 shows an oscilloscope presentation, a precision photo mask, and a microscope view of a fabricated circuit for one metalization pattern from this memory. This written facility is also being used by a service group in the Laboratory. They are now able to undertake some tasks previously impractical with conventional rubylith pattern-making techniques. Creating the host program was a two day job due to the conveniences of LEAP and the fact that the tape-punching program already had been written for the sketching layout program.

C. Printed Circuit Card Layout

Large-scale integrated circuit technology (LSI) shows promise of giving low cost, fast hardware, but poses new design problems. To focus on these problems, the design of an LSI digital processor has been undertaken. Of major interest is the use of our existing computer and graphics facilities to aid in the design. In particular, we have constructed programs for aiding LSI mask generation (see previous section) and for making printed circuit card masks.

The philosophy behind the two programs is similar. The computer and graphics facility is primarily a drafting aid. The user draws a representation of the mask using the Sylvania Data Tablet, and simple character recognition programs translate the user's handwriting into data meaningful to the computer. For making printed circuit card masks, the user merely connects together with straight line segments a set of points which represent component contacts. Component contacts and lines may be added, moved, or deleted, and finally a paper tape from which the masks are made is punched by the TX-2.

Further experimentation is necessary before a useful design tool can be constructed. For example, we envision a man/machine interactive facility as opposed to a total computer solution to the wire routing and component placement problem. The man could select the order in which the wires were considered for routing, but the computer could find the best path.

D. TIC (Testing Integrated Circuits) Terminal

Our initial effort toward the automation of logic circuit testing, as reported in the previous Semiannual Technical Summary,[2] resulted in the construction of the TIC Terminal on the TX-2. and the implementation of the TIC programming language. The terminal will now allow digitized voltages as well as logic values to be input to TX-2. The language and terminal have been used successfully to test several hundred micrologic memory circuits. Recently, the features of the TIC language peculiar to operating the TIC terminal have been added to LEAP thereby obsoleting the TIC language. The TIC language was simple and had good debugging facilities, but lacked many useful language constructs. The VITAL compiler-compiler system has made modifications of this sort straightforward and painless.

E. Diagnosis of Single Gate Failures in On-Line Testing

The most interesting and important results have arisen from a study of computer-generated diagnostic tests. We were especially interested in theory and methods applicable to LSI circuits, i.e., circuits with more than 50 gates in which most gate outputs cannot be directly probed. To simplify testing and to satisfy our own needs, only combinational circuits were considered. The literature, unfortunately, contains no satisfactory testing methods for the broad class of failures we wished to diagnose.

The study resulted in the development of diagnostic test procedures based on a single gate failure hypothesis. We postulate that a logic circuit can be partitioned into a set of gates where a gate is defined as any multiple input, single output combinational circuit, and further, that a failure is due to an arbitrary transformation of the correct function of any one, but only one, of the gates. A test is the application of one input combination to the input terminals and the interpretation of the response at one of the output terminals. We have developed methods for finding test sets which will detect any (detectable) single gate failure, identify the offending gate (or gates, since certain failures cause equivalent terminal bahavior), and specify the function actually being generated by the faulty gate.

The bounds which have been derived for the size of the test sets are encouragingly small. If it is assumed the circuit consists of K gates, each with m inputs, at most $K2^m + K - 1$ tests need to be applied. An alternative but equivalent procedure requiring less computation at the time of testing requires storage for $K^2 2^m$ tests although fewer than $(K - 1)2^{m+1}$ must be applied to the circuit being tested. The latter scheme is being implemented on the TX-2. A paper describing this new approach to testing has been submitted for publication.

F. Buffered TV-Scan Display

Plans are being made to provide a hardware interface which will allow the one-million-bit LCM memory module to be used as a buffer store for a flicker-free TV-scan type display. The initial installation will provide primitive scan conversion for loading the memory a point at a time.

G. Conic Generator

We are now engaged in an effort to substantially increase the drawing speed of the TX-2 conic generator.[3,4] The most critical components of the conic generator are the T and T^2 multiplying D/A converters, identical devices which increment the time parameter during the drawing of a line or curve segment. The converters are driven by a counter whose counting rate determines the speed with which a segment is drawn. Spikes in the otherwise smooth analog outputs of the converters occur when the counter changes state. At relatively low counting (drawing) rates, the effect of the spikes on the display is not visible, but the spikes become more objectionable as the rate is increased. The switching spikes presently place an upper limit on a drawing speed of 1 msec for a full screen segment. This corresponds to a counting rate of 1 MHz. For display frames with many picture elements, typical of circuit card layout and integrated circuit mask design, the resulting flicker is excessive. Our effort during the past months has been toward the development of a quieter multiplying D/A converter. A converter has been designed and built which has far less switching noise. Its use in the conic generator will allow a substantial increase in drawing speed. The main features of the new design are:

(1) A four times increase in signal level within the converter.

(2) Inclusion of the counter as an integral part of each converter assembly (presently one counter is shared by both T and T^2 converters and connected to them via back panel wiring).

(3) Use of a new high-speed integrated logic family (MECL II), reducing bit to bit counter skew (a serious source of noise) to 1 nsec, and providing logic swings precisely tailored to the converter requirements.

(4) Completely new packaging to assure good separation between digital signals and sensitive analog leads.

Bench tests of the new converter showed noise spike amplitudes reduced by 60 times with respect to the present noise levels. Operation at a counting rate of 30 MHz seems as quiet as our present converter operating at a counting rate of 1 MHz. It remains to be seen how much system performance will be improved when the new D/A converters are installed in the conic generator early this summer.

H. Tablet Editor

Many interactive computer applications on TX-2 require, or would benefit from, a capability of recognizing symbols drawn by the user on the tablet. The symbols might be characters (for test editing, picture labelling, or on-line input), picture components (for circuit drawing or logic design), picture manipulation commands (deleting or moving parts of a picture), or program commands (instead of light targets). A symbol recognition routine is being developed to make these facilities available to any program that desires them. This note presents some thoughts on the design of such a program.

The variety of symbols required by the many possible applications and the large number of potential users indicate that the program should be trained by each user to recognize his own versions of whatever symbols he requires. The alternative approach of producing a fixed standard symbol set would probably be more reliable and more efficient, since many problems could be handled by special techniques. However, many users would have to adapt to the standard symbol set (this is particularly undesirable for characters), and adding symbols for new applications would involve reprogramming the recognizer.

The raw data points obtained by sampling the tablet should first be smoothed and thinned. At the same time information about slope changes and extreme points can be obtained if these are used in the recognition technique. The preprocessing phase should ideally be done as the points are being sampled; however, since the recognizer will operate under time sharing, this is not possible on TX-2, since the sampling rate must be on the order of a few milliseconds.

Most of the work in symbol recognition is concerned with the analysis of single strokes. The idea is to extract from the data points enough features independent of size and position to distinguish "different" strokes while at the same time minimizing the number of versions of "identical" strokes which must be given to the recognizer. The gross "shape" of a stroke can be described by such features as the sequence of (quantized) changes of direction, location of local extrema, or the path through elements of a grid superimposed on the stroke. In addition, some detailed features must be recorded, such as curvature to distinguish boxes from circles, terminal point location to distinguish a check "√" from a "V", and aspect ratio to distinguish a "C" from a left parenthesis "(". The different types of features should be recorded separately, since they may not all be needed for a given user's symbol set. For example, if the user does not want "√" and "V" to be distinguished, he should only have to enter one of the symbols; the recognizer should be able to determine that "√" is similar enough to "V" to say that they are identical unless it is told that they are different. The same considerations apply to shape-independent features such as the direction in which a stroke is drawn (clockwise versus counterclockwise circles, or vertical lines drawn top to bottom or bottom to top), and to the order in which the strokes of a multiple-stroke character are drawn. The reliability and usefulness of a symbol recognizer depend heavily on how well it performs these "closeness" evaluations.

Certain types of symbols involve strokes which are too complex to be described as a single entity; the most common example is the lower case script alphabet. A special purpose recognizer could probably handle these symbols with techniques such as eliminating "tails" and recognizing "loops"; but they are difficult symbols for a general-purpose recognizer. One possible technique would be to detect when a stroke becomes too complex for a single descriptor and handle it as a multi-stroke character, breaking strokes at sharp corners, inflection points, or crossovers.

Multiple-stroke character descriptions require two types of information: the descriptions of their component strokes, and the positional relationship between these strokes. The relative position of strokes can be recorded either by the locations of the centers on a grid, or by encoding whether one stroke is above, below, left, right, or coincident, with respect to another.

The dictionary structure is partially dependent on the encoding used for the various feature types of a stroke and the representation of multi-stroke characters, and partially on the technique used to optimize search time (tree-structure, hash-coding, or sorting, for example). It is clear that some good search technique should be used, since the dictionary is searched much more often than it is modified.

The output of the recognizer should include the symbol identifier (if recognized), the center and dimensions of the minimum rectangle fitted about the character, and the smoothed data point.

The training program should be able to display for any entry a stylized version of the symbol which reflects its internal representation. This will allow the user to resolve any ambiguities which arise while he is constructing the dictionary. For example, if he tries to enter a check "√", and discovers that this is already defined as a "V", he can determine whether his problem is in drawing the check incorrectly or in having inadvertently defined his "V" with the left side too short. This feature will also help the user who is having trouble discovering what he is doing wrong.

The design philosophy of a recognizer changes radically if a picture-drawing is considered. For example, the recognizer described above is not capable of recognizing such "symbols" as "arrows" drawn in various orientations. Perhaps what is needed for this type of application is a program which classifies strokes as one of a number of simple fixed shapes (lines, circles, cups, boxes, etc.), and outputs the shape, the orientation, and the significant points. More thought needs to be given to this area.

A preliminary version of a symbol recognizer has been implemented using a technique somewhat different from any of those described above. The shape of the strokes in a symbol is described by encoding its behavior in the x- and y-axes separately; if the descriptor for one of the axes matches an entry exactly and the descriptor for the other is "close", the shape is considered to be matched. Aspect ratio and the appearance of corners are used to decide between symbols having identical shape descriptors. The recognizer is currently being used in a text editor which is under development, and results so far have been excellent. However, this scheme has its peculiarities and limitations, and must be augmented by some of the previously described techniques in order to be useful as a general-purpose recognizer.

I. Constraint Systems

1. Introduction

During the last quarter, we have focused our attention on large, sparse linear systems, such as those arising from difference equations. We have concentrated on such systems because they have wide applicability and because our earlier work leads us to expect that results will be generalizable.

In general, Gaussian elimination methods are inappropriate for large sparse systems, since inordinately many nonzero coefficients can be introduced, presenting storage and speed problems. Therefore, the approaches generally taken are iterative, such as the Gauss-Seidel method or the Jacobi method. However, iterative methods have definite speed limitations.

In contrast to these number-crunching techniques, our approach has been to represent linear systems by a geometric structure and to study the structural properties revealed by such a representation. For example, the connectivity of the structure reflects the sparseness of coefficients. Since our solution techniques to date are based on propagation of values through the structure, it is not surprising that connectivity or sparseness studies have a direct bearing on optimizing the efficiency of finding solutions.

2. Present Progress

Progress has been made in two complementary areas: (a) solution procedure, and (b) understanding the relation between sparseness and optimization of the solution procedure.

(a) The effect of our solution procedure is to accomplish for large, sparse systems what Gaussian elimination does for small, dense systems. The original system, characterized by a high degree of interdependence of variables, is transformed into a system whose variables are cascaded. The interdependencies are untangled, so to speak.

Of course, the critical issue for applications is: what price has to be paid to obtain this simplified system? For example, Gaussian elimination can accomplish the same result; however, as noted above, the storage requirements are often prohibitive.

In our recent work, we have developed a simplification method that is undemanding in terms of both storage and logic. Utilizing the connectivity of the constraint structure, we have shown that the simplified "cascade" structure itself can be generated by a simple cascade process.

(b) Roughly speaking, this cascade process can proceed along more than one path through the structure. The solution will be the same regardless of the choice of path; however, the speed of solution will vary with the choice. We have attempted to understand how to optimize this choice.

It has been found that there is quite a direct relationship between optimal choice and the distribution of nonzero coefficients. Though work is not yet completed, it is proceeding encouragingly in the direction of an optimal choice-minimal cover theorem. The reader may recall the use of minimal covers in switching and automata theory.

3. Summary

Among the possible benefits of these developments, we will discuss three that presently interest us. First, our methods achieve a substantial increase in speed over earlier iterative methods. This increase may make possible simulation studies that were formerly unfeasible. The reason for this speed increase is that our methods lead to analytic rather than iterative solutions. The cascade process mentioned above eliminates parsing difficulties and minimizes the complexity of compiling a "solution subroutine." This subroutine is a piece of code whose input is the set of nonzero coefficients and whose output is the value of the unknown variables. With such a subroutine, the effects of modifying the nonzero coefficients can easily be simulated, permitting interactive variational studies that are beyond the present scope of a purely mathematical analysis.

Second, the framework of these studies is a natural one in which to examine applications of parallel processing. In the course of our work, ideas have recurred which appear capable of supporting fruitful investigations into parallelism. Among these ideas are those of cascading along parallel paths in a constraint structure, and "factoring" a structure into pieces.

Lastly, the mathematical framework itself offers a fresh viewpoint toward numerical problems. Traditionally, such problems are studied in terms of the concept of algorithm and related ideas, as opposed to the concept of structure utilized in much of mathematical research. To be sure, the automaton concept has been revealed to be in effect a "dual" of the algorithm concept, and the structural theory of automata has been intensively developed. However, these studies have had little effect on numerical analysis. On the other hand, numerical and many other relationships have an inherent algebraic structure quite distinct from the structural concepts of automata theory. Our work attempts to show that this inherent structure of constraints can be studied with practical benefit.

J. Waveform Processing

This project involving the multiplicative filtering of images[5] has four basic phases: fundamental digital photography, simultaneous contrast enhancement and dynamic range reduction, bandwidth reduction, and psychophysical experimentation. Project progress and results are summarized by phase.

1. Phase A

The scanner used to input images for this project is a very inexpensive mechanical facsimile device. Its disadvantages are slow speed (600 picture elements per second), fixed raster size (96 samples per inch), and input medium (glossy photographic prints). Its chief advantages are extremely low noise, excellent black level control, and high dynamic range. These characteristics have been obtained by using strong illumination, well regulated power supplies, and standard noise reduction practice. The last increment in black level control was obtained by compensating for tiny variations in the frequency response of the scanner amplifier circuits by using suitable difference equations on the digitized picture signal. Dynamic range beyond the contrast of printing papers was successfully obtained by printing a standard gray scale next to the negative and providing an algorithm for deriving the appropriate nonlinear density compensating curve. By using high latitude printing papers, information from large dynamic range negatives can be transferred faithfully to the computer.

A new display generator was installed on TX-2, and the image output programs had to be redesigned while special changes were made to the equipment. The results are improved continuous tone digital photographic output at increased speed. Typical 8-bit images consisting of 340 x 340 picture elements can be output in approximately 15 seconds onto any wide latitude film. Emulsion nonlinearities are conveniently compensated producing high grayscale fidelity with any medium such as Polaroid or standard negative films.

2. Phase B

Contrast enhancement combined with dynamic range reduction is illustrated in Figs. 6 and 7. Figure 6 is an unprocessed image and Fig. 7 has been processed for lower apparent illumination variation and increased sharpness. The details of this processing were discussed previously.[5]

An image was processed for simultaneous dynamic range reduction and contrast enhancement using a one-dimensional filter. The results were surprisingly good and imply that a real time system might be built using conventional television equipment, electric network filters, and logarithmic amplifiers.

Enhancement processing has been applied to X-ray images. Photographic gammas as high as 32 were applied to the sharpened components with the result that obscure detail in soft tissue was rendered readily visible. A series of lung X-rays was filtered in an effort to enhance the visibility of a growing tumor plainly visible only in the last of the series. The results await interpretation by the medical community.

3. Phase C

A study of the periodograms of digital images and their logarithms was performed. One objective was to determine an appropriate whitening filter for typical images under the assumption that whitening is an appropriate precursor to bandwidth reduction processing. Investigations are still under way.

4. Phase D

Preliminary efforts to cancel the optical illusion known as Mach-bands by means of inverse filtering produced weak positive results on the basis of two trials. To expedite explorations, extensive planning, design, and coding for a convenient space and frequency domain filter synthesis program was performed.

The specific details of the image processing project will be presented in the IEEE Proceedings[6] along with other related topics concerning nonlinear waveform processing.

K. BMRS

The Ballistic Missile Re-entry System project is a consulting effort in the area of man-machine communications. The task consists of a detailed analysis of the procedures and practices used in the editing of radar plots. The approaches developed are tested on the TX-2 time-sharing system. At each step in development, the potential users evaluate and comment on the results to date. The present BMRS editor program has the ability to handle the plotting of radar plot data generated on the IBM 360 Model 67 and presents the data on a TX-2 CRT display. The features of this editor include:

(1) The conversion of 360 characters and floating point numbers to TX-2 characters and floating point numbers using a fast conversion algorithm developed at TX-2.

(2) The options of plotting in point, line, and character mode.

(3) The ability to plot as many as four separate plots on the screen at one time. In addition to these features, the BMRS editor has the rudimentary editing features of plot selection by frame number and the ability to go backward or forward from any given plot under typewriter control.

Changing the rudimentary editing controls to tablet control is under way. This change seems to be an easy way to find out what controls are needed and how to present the controls easily to the user.

The BMRS data base consists of strings of radar cross section plots and their associated Trade Plots. The amount of data involved for one test can be as many as 30,000 plots. The large volume of data demands that the method of selection be very fast and easy to use. In this area, we are setting up two levels of selection tables. The method of using these tables and the information content will be the result of further research.

Since the final product of editing is another plot of values which are taken from a given string of radar cross sections, the ability to build this final plot as one edits the data is planned. The problems of selection, storage, and presentation have not been solved, and this is one of the more challenging parts of this work.

The final phase of study will be the modification of the cross section plot during editing. Since the computation involved is beyond the scope of TX-2, the recalculation will be simulated for test purposes only. Here again the problems of selection, recalculation, storage, wait time and presentation have to be considered.

The BMRS project will take the combined knowledge of TX-2 and apply it to a very real and challenging applications problem. The results of this work will be used in the design of future systems for the IBM 360 Model 67 by the BMRS group.

ACKNOWLEDGMENT

Special recognition is due Martha Greenberg from Harvard and D. Austin Henderson, Jr. from M.I.T. for their efforts in creating the AMBIT/G system. Miss Greenberg worked on the problem of displaying an output AMBIT/G data graph, and Mr. Henderson wrote the interpreter and helped with the output problem.

REFERENCES

1. C. Christensen, "An Example of the Manipulation of Directed Graphs in the AMBIT/G Programming Language," Proceedings of the Symposium on Interactive Systems for Experimental Applied Mathematics, Washington D.C., 26 – 28 August 1967.
2. Semiannual Technical Summary to the Advanced Research Projects Agency on Graphics, Lincoln Laboratory, M.I.T. (30 November 1967), DDC 663728.
3. L.G. Roberts, "Conic Display Generator Using Multiplying Digital-Analog Converters," Trans. IEEE Elec. Computers EC-16 (3), 423 (June 1967).
4. H. Blatt, "Conic Display Generator Using Multiplying Digital-Analog Decoders," Proc. of the Fall Joint Computer Conference, p. 177 (1967).
5. Graphics Semiannual Technical Summary, op. cit., pp. 20 – 24.
6. A.V. Oppenheim, R.W. Schafer and T.G. Stockham, "The Nonlinear Filtering of Multiplied and Convolved Signals," Proc. IEEE, to be published.

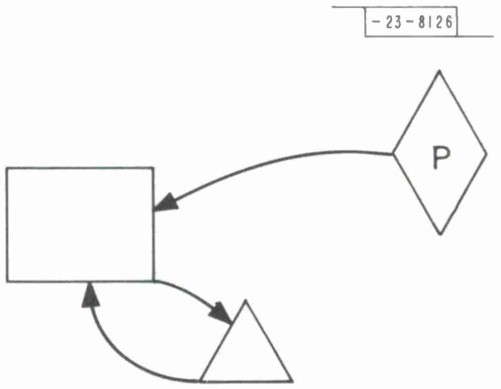
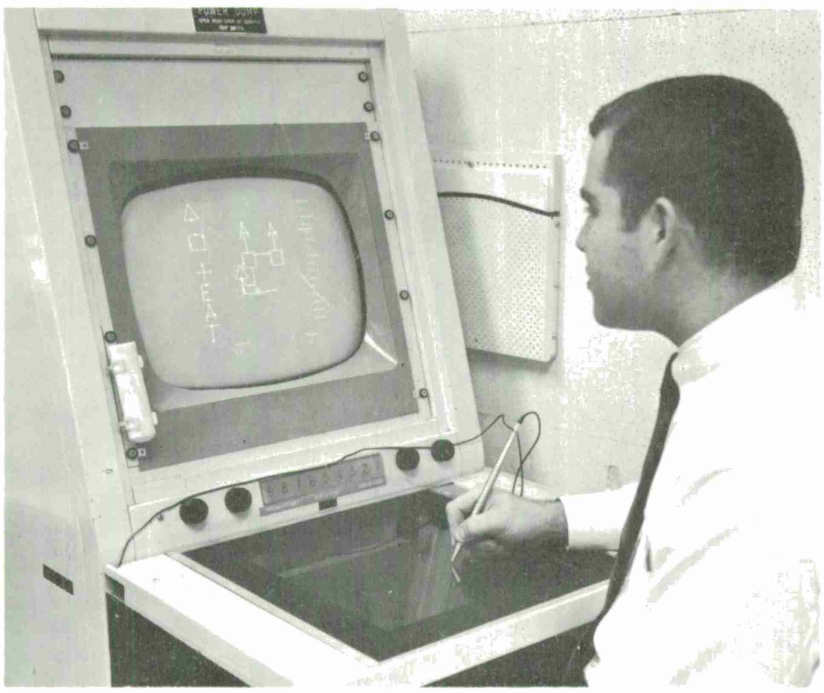
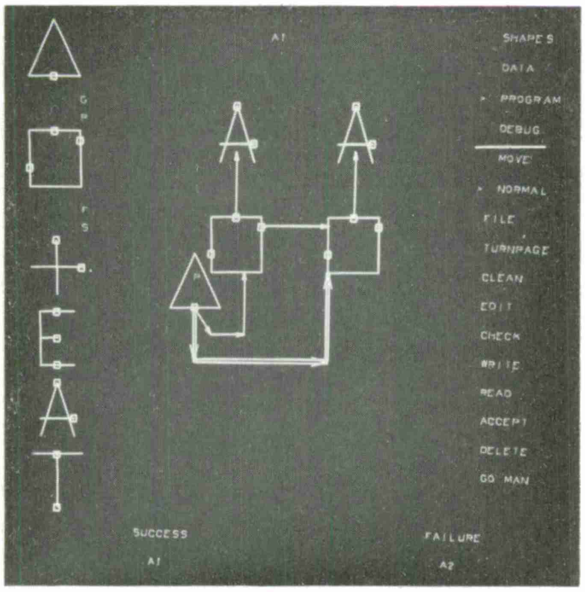


Fig. 1. Example of an AMBIT/G data graph.

-23-8127



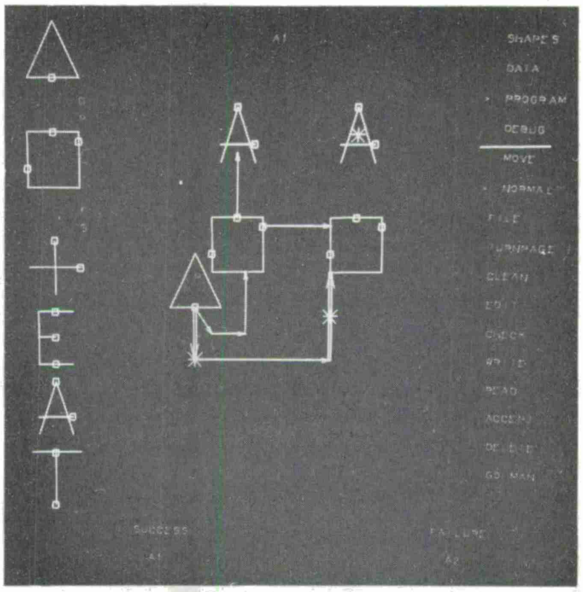
(a)



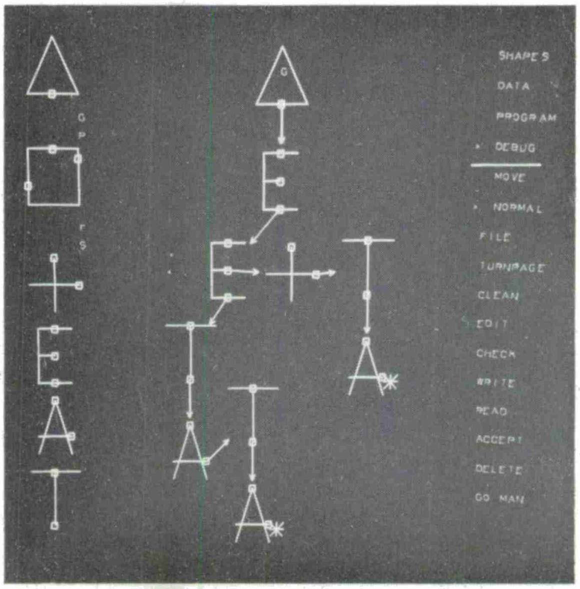
(b)

Fig. 2. (a) Console display of AMBIT/G program statement input. (b) Typical AMBIT/G program statement as it appears in the working area.

-23-8128

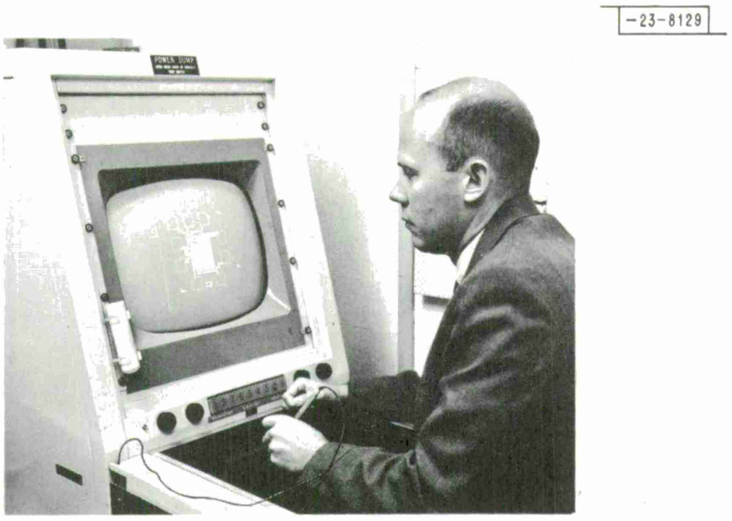


(a)

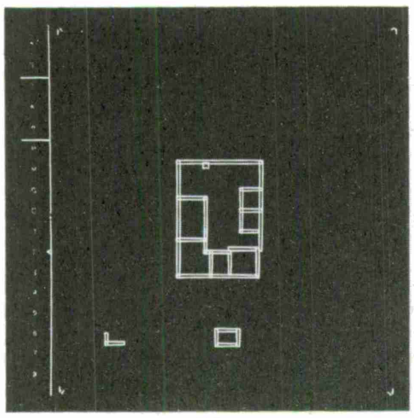


(b)

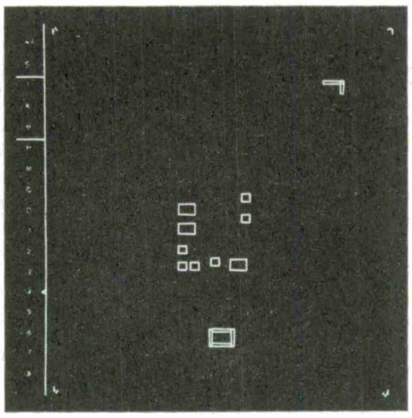
Fig. 3. (a) AMBIT/G program statement with syntax errors; an illegal sequence of lines, a dangling line and an unreachable node (each error is indicated by a displayed star). (b) Typical AMBIT/G output data graph representing a simple computation tree.



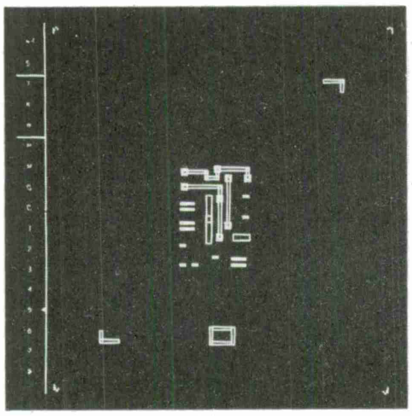
(a)



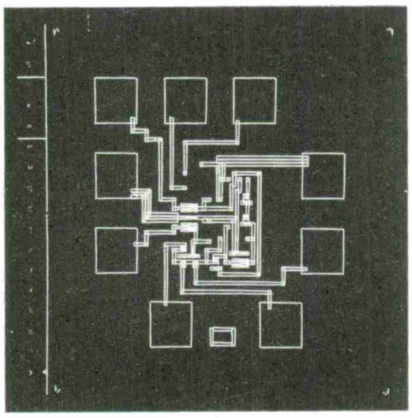
(b)



(c)

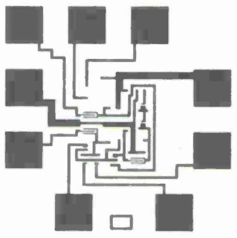


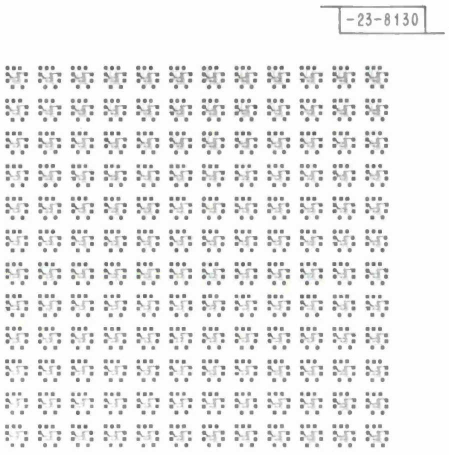
(d)



(e)

Fig. 4. Semiconductor mask design. (a) Console display of circuit layout. (b) Isolation diffusion pattern. (c) Transistor base diffusion pattern. (d) Resistor and base insert diffusion pattern. (e) Metallization pattern.


(f)



(g)

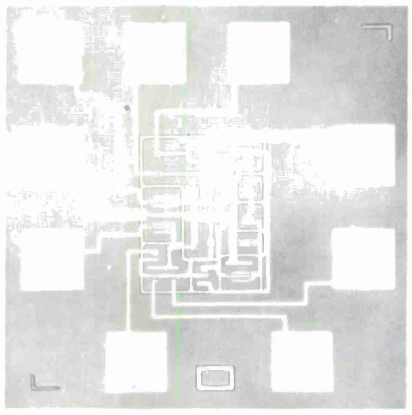
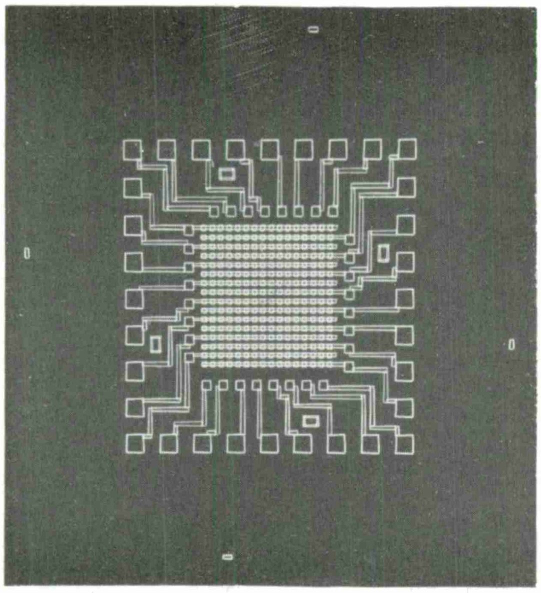
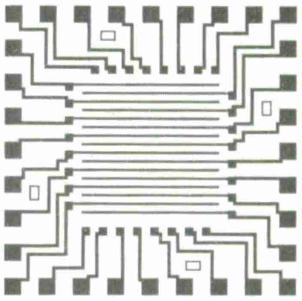

(h)

Fig. 4. Continued. (f) Enlarged view of metallization precision pattern (pads are 30 mils square actual size). (g) Enlarged view of stepped metal pattern used for fabrication (pads are 3 mils square actual size). (h) Microscope view of fabricated circuit.

-23-8131



(a)



(b)

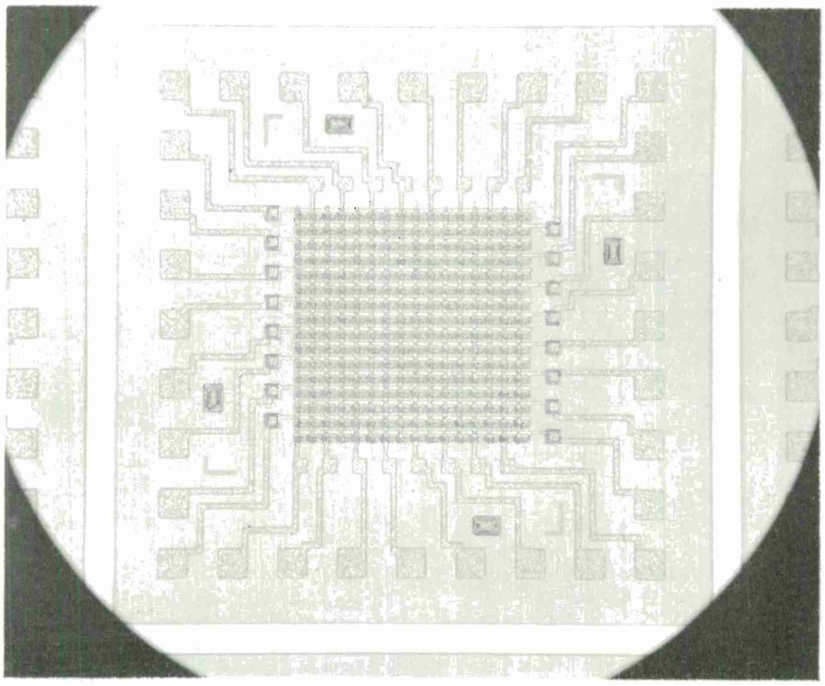


Fig. 5. Read-only memory patterns. (a) Scope view of second level metal pattern. (b) Precision second level metal pattern. (c) Microscope view of fabricated circuit.

-23-8132

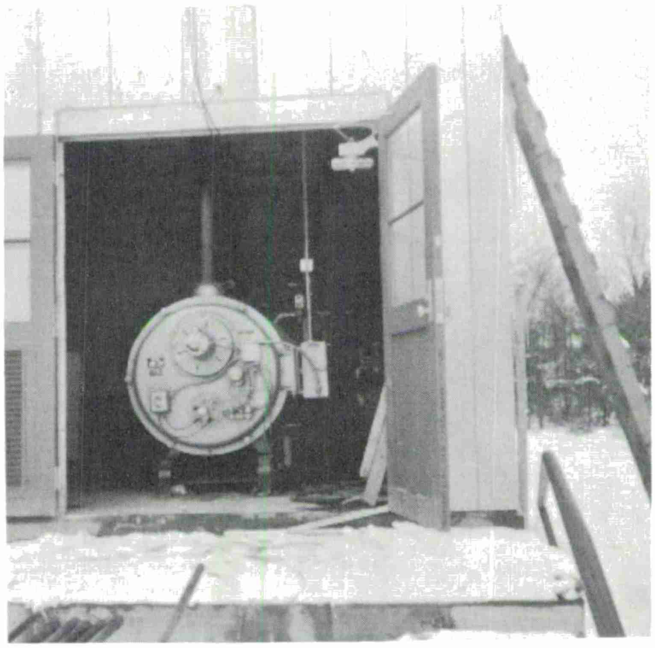


Fig. 6. Unprocessed image.

-23-8133

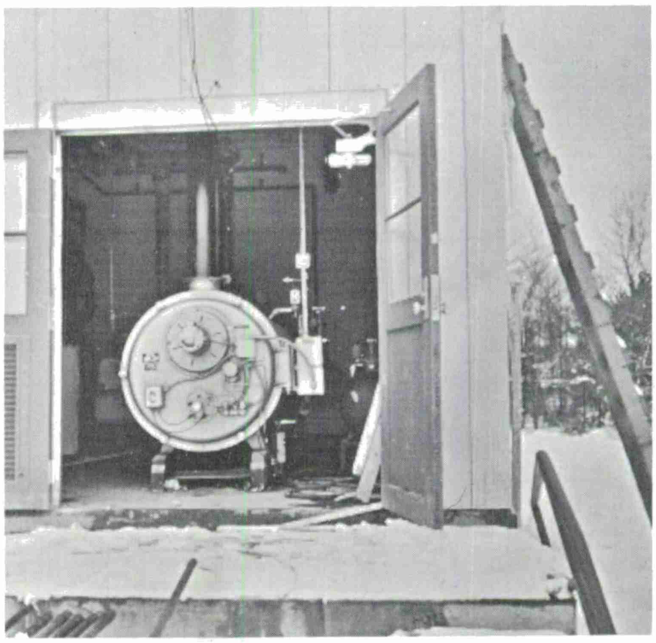


Fig. 7. Processed image.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA – R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Lincoln Laboratory, M.I.T.	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP None
3. REPORT TITLE Semiannual Technical Summary Report to the Advanced Research Projects Agency for Graphics	
4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)* Semiannual Technical Summary Report (1 December 1967 through 31 May 1968)	
5. AUTHOR(S) *(Last name, first name, initial)* Raffel, Jack I.	

6. REPORT DATE 31 May 1968	7a. TOTAL NO. OF PAGES 32	7b. NO. OF REFS 6

8a. CONTRACT OR GRANT NO. AF 19(628)-5167	9a. ORIGINATOR'S REPORT NUMBER(S) Semiannual Technical Summary for 31 May 1968
b. PROJECT NO. ARPA Order 691	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* ESD-TR-68-61
c.	
d.	
10. AVAILABILITY/LIMITATION NOTICES This document has been approved for public release and sale; its distribution is unlimited.	
11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

The LEAP system has been modified to provide for merging current and previously saved data structures and incorporation of a sublanguage for communicating with the interrupt executive. Breakpoint trapping facilities have been used for running programs in single-step mode, program-timing experiments, and in providing variable response delays for human factors studies.

AMBIT/G, a programming language for manipulating directed graphs, is being implemented using LEAP and the interactive graphics facilities of TX-2.

A first set of working semiconductor circuits has been made from masks generated via TX-2 programs. A written-input version of the mask-generation program has been used to produce masks for a Read-Only Memory design. In addition, the laboratory service facility is now using this program for almost all of its hybrid circuit mask layout jobs.

Work has continued on computer-aided circuit testing with the addition of new features to the TIC terminal and the development of procedures for diagnosing single-gate failures in complex arrays.

Experiments are planned for evaluating a TV display buffered by the film memory soon to be installed in TX-2. Modifications to the conic generator will be made shortly which should eliminate the generator limitations on drawing speed.

Further theoretical work continues on constraint systems and the application of homomorphic filtering to picture processing.

14. KEY WORDS

graphical communication	man-machine
TX-2	display systems
time-sharing	programming languages

UNCLASSIFIED

Security Classification

Printed by
United States Air Force
L. G. Hanscom Field
Bedford, Massachusetts